A dynamic view of active integrity constraints

Guillaume Feuillade, Andreas Herzig

A Dynamic View of Active Integrity Constraints

Guillaume Feuillade and Andreas Herzig

Université de Toulouse, IRIT-LILaC and CNRS, France

Abstract. Active integrity constraints have been introduced in the database community as a way to restore integrity. We view active integrity constraints as programs of Dynamic Logic of Propositional Assignments DL-PA and show how several semantics of database repair that were proposed in the literature can be characterised by DL-PA formulas. We moreover propose a new definition of repair. For all these definitions we provide DL-PA counterparts of decision problems such as the existence of a repair or the existence of a unique repair.

Keywords: Active integrity constraints, dynamic logic, propositional assignments.

1 Introduction

Updates under integrity constraints is an important and notoriously difficult issue in databases and AI. About ten years ago, active integrity constraints were proposed in the database literature as a 'more informed' way of maintaining database integrity [FGZ04, CTZ07, CT08, CGZ09, CT11, CF14]. There, an active integrity constraint is basically viewed as a couple $r = \langle \mathsf{C}(r), \mathsf{R}(r)\rangle$ where $\mathsf{C}(r)$ is a formula and $\mathsf{R}(r)$ is a set of update actions each of which is of the form either $p{\leftarrow}\top$ or $p{\leftarrow}\bot$, for some atomic formula p. The idea is that (1) when $\mathsf{C}(r)$ is true then the constraint r is violated, and (2) a violated constraint can only be repaired by performing one or more of the update actions in $\mathsf{R}(r)$.

In this paper we examine active integrity constraints in the framework of dynamic logic and argue that they should be viewed as a complex program: the sequential composition of the test of $\mathsf{C}(r)$ and the nondeterministic choice of an action in $\mathsf{R}(r)$. Repairing a database can then be done by means of a complex program that combines active integrity constraints. We use a simple yet powerful dialect of dynamic logic: Dynamic Logic of Propositional Assignments, abbreviated DL-PA [HLMT11, BHT13]. The latter is a simple instantiation of Propositional Dynamic Logic PDL [Har84, HKT00]: instead of PDL's abstract atomic programs, its atomic programs are update actions: assignments of propositional variables to either true or false, written $p{\leftarrow}\top$ and $p{\leftarrow}\bot$. Just as in PDL, these atomic programs can be combined by means of program operators: sequential and nondeterministic composition, finite iteration, and test. While DL-PA programs describe the evolution of the world, DL-PA formulas describe the state of the world. In particular, formulas of the form $\langle\pi\rangle\varphi$ express that φ is true after *some* possible execution of π, and $[\pi]\varphi$ expresses that φ is true after *every* possible execution of π. The models of DL-PA are considerably simpler than PDL's Kripke models: valuations of classical propositional logic are enough. The assignment $p{\leftarrow}\top$ inserts p, while the assignment $p{\leftarrow}\bot$ deletes p. It is shown in [HLMT11, BHT13] that every DL-PA formula can be reduced

to an equivalent boolean formula. This will allow us to construct repaired databases syntactically.

Just as [CT11, CF14] we only consider ground constraints, i.e., we work with a propositional language.

The paper is organized as follows. After some preliminaries (Section 2) we recall DL-PA in Section 3. In Section 4 we recall static constraints and provide an embedding of the associated repairs that have been defined in the literature into DL-PA. In Section 5 we do the same for active integrity constraints. In Section 6 we propose a new definition in terms of **while** programs. Section 7 concludes.

2 Preliminaries

In this paper we consider propositional languages that are built from a countable set of propositional variables (alias atomic formulas) $\mathbb{P} = \{p, q, \ldots\}$. *Boolean formulas* are built from $\mathbb{P}$ by means of the boolean operators $\top$, $\bot$, $\neg$, and $\vee$ and are denoted by A, B, etc. The other boolean connectives $\wedge$, $\rightarrow$, and $\leftrightarrow$ are abbreviated in the usual way. A *literal* is an element of $\mathbb{P}$ or the negation of an element of $\mathbb{P}$ and a *clause* is a disjunction of literals. We define $\mathbb{P}_A$ to be the set of variables from $\mathbb{P}$ occurring in formula A. This extends to sets in the obvious way.

Valuations are subsets of $\mathbb{P}$ and are denoted by V, V_1, V_2, etc. The set of all valuations is therefore $\mathbb{V} = 2^{\mathbb{P}}$. It will sometimes be convenient to write $V(p) = \top$ instead of $p \in V$ and $V(p) = \bot$ instead of $p \notin V$. In the context of active integrity constraints a valuation is called a *database*.

A valuation determines the truth value of every boolean formula. The set of valuations where A is true is noted $\|A\|$. We sometimes write $V \models A$ when $A \in \|V\|$.

An *update action* is of the form $p{\leftarrow}\top$ and $p{\leftarrow}\bot$, for $p \in \mathbb{P}$. The former is the insertion of p and the latter is the deletion of p. We denote the set of all update actions by $\mathbb{U}$. We sometimes use X as a metavariable for $\top$ and $\bot$ and write $p{\leftarrow}\mathsf{X}$. For subsets P of $\mathbb{P}$ it will be convenient to write $P{\leftarrow}\top$ to denote the set of update actions $\{p{\leftarrow}\top \ : \ p \in P\}$, and likewise for $P{\leftarrow}\bot$. A set of update actions $U \subseteq \mathbb{U}$ is *consistent* if it does not contain both $p{\leftarrow}\top$ and $p{\leftarrow}\bot$, for some p.

The *update* of a valuation V by a set of update actions U is defined as:

$$V{\circ}U = (V \setminus \{p \ : \ p{\leftarrow}\bot \in U\}) \cup \{p \ : \ p{\leftarrow}\top \in U\}$$

So all the deletions are applied in parallel first, followed by the parallel application of all insertions. We could as well have chosen some other order of application. When U is consistent then all of them lead to the same result. In particular:

Proposition 1. *Let $\{\alpha_1, \ldots, \alpha_n\}$ be a consistent set of update actions. Let $\langle k_1 \ldots k_n \rangle$ be some permutation of $\langle 1 \ldots n \rangle$. Then $V{\circ}\{\alpha_1, \ldots, \alpha_n\} = (\ldots(V{\circ}\{\alpha_{k_1}\})\ldots){\circ}\{\alpha_{k_n}\}$.*

3 Dynamic Logic of Propositional Assignments

The first studies of assignments in the context of dynamic logic are due, among others, to Tiomkin and Makowski and van Eijck [TM85, vE00]. Dynamic Logic of Propositional Assignments DL-PA was introduced in [HLMT11] and was further studied

in [BHT13]. Evidence for its widespread applicability was provided in several recent publications, including belief update and belief revision, argumentation, and planning [Her14, DHP14, HMNDBW14]. We briefly recall syntax and semantics.

3.1 Language

The language of DL-PA is defined by the following grammar:

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi$$
$$\pi ::= \alpha \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid \pi^- \mid \varphi?$$

where p ranges over the set of atomic formulas $\mathbb{P}$ and α ranges over the set of update actions $\mathbb{U}$. In DL-PA, update actions are called atomic assignments. The operators of sequential composition (";"), nondeterministic composition ("$\cup$"), finite iteration ("$(.)^*$", the so-called Kleene star), and test ("$(.)?$") are familiar from PDL. The operator "$(.)^-$" is the converse operator. The formula $\langle\pi\rangle\varphi$ is read "there is an execution of π after which φ". The star-free fragment of DL-PA is the subset of the language made up of formulas without the Kleene star "$(.)^*$".

We define $\mathbb{P}_\varphi$ to be the set of variables from $\mathbb{P}$ occurring in formula φ, and we define $\mathbb{P}_\pi$ to be the set of variables from $\mathbb{P}$ occurring in program π. For example, $\mathbb{P}_{p\leftarrow q \cup p\leftarrow\neg q} = \{p, q\} = \mathbb{P}_{\langle p\leftarrow\bot\rangle q}$.

Several program abbreviations are familiar from PDL. First, **skip** abbreviates $\top?$ and **fail** abbreviates $\bot?$. Second, **if** φ **then** π_1 **else** π_2 is expressed by $(\varphi?;\pi_1) \cup (\neg\varphi?;\pi_2)$. Third, the loop **while** φ **do** π is expressed by $(\varphi?;\pi)^*;\neg\varphi?$. Let us moreover introduce assignments of literals to variables by means of the following two abbreviations:

$$p\leftarrow q = \textbf{if } q \textbf{ then } p\leftarrow\top \textbf{ else } p\leftarrow\bot \qquad p\leftarrow\neg q = \textbf{if } q \textbf{ then } p\leftarrow\bot \textbf{ else } p\leftarrow\top$$

The former assigns to p the truth value of q, while the latter assigns to p the truth value of $\neg q$. In particular, the program $p\leftarrow\neg p$ flips the truth value of p. Note that both abbreviations have constant length, namely 14. Finally and as usual in modal logic, $[\pi]\varphi$ abbreviates $\neg\langle\pi\rangle\neg\varphi$.

3.2 Semantics

DL-PA programs are interpreted by means of a relation between valuations. The atomic programs α update valuations just as singleton sets of update actions do (cf. the preceding section), and complex programs are interpreted just as in PDL by mutual recursion. Table 1 gives the interpretation of formulas and programs. where $\circ$ is relation composition and $(.)^{-1}$ is relation inverse.

A formula φ is DL-PA *valid* iff $\|\varphi\| = 2^{\mathbb{P}} = \mathbb{V}$. It is DL-PA *satisfiable* iff $\|\varphi\| \neq \emptyset$. For example, the formula $\langle p\leftarrow\bot\rangle\top$, $\langle p\leftarrow\top\rangle\varphi \leftrightarrow \neg\langle p\leftarrow\top\rangle\neg\varphi$, $\langle p\leftarrow\top\rangle p$, and $\langle p\leftarrow\bot\rangle\neg p$ are all valid.

Observe that if p does not occur in φ then formulas such as $\varphi \rightarrow \langle p\leftarrow\top\rangle\varphi$ and $\varphi \rightarrow \langle p\leftarrow\bot\rangle\varphi$ are valid. This is due to the following semantical property that is instrumental in the proof of several results in the rest of the paper.

Table 1. Interpretation of formulas and programs

$\|\|p\|\| = \{V : p \in V\}$	$\|\|\alpha\|\| = \{\langle V_1, V_2\rangle : V_2 = V_1 \circ \{\alpha\}\}$
$\|\|\top\|\| = \mathbb{V} = 2^{\mathbb{P}}$	$\|\|\pi;\pi'\|\| = \|\|\pi\|\| \circ \|\|\pi'\|\|$
$\|\|\bot\|\| = \emptyset$	$\|\|\pi \cup \pi'\|\| = \|\|\pi\|\| \cup \|\|\pi'\|\|$
$\|\|\neg\varphi\|\| = 2^{\mathbb{P}} \setminus \|\|\varphi\|\|$	$\|\|\pi^*\|\| = (\|\|\pi\|\|)^*$
$\|\|\varphi \vee \psi\|\| = \|\|\varphi\|\| \cup \|\|\psi\|\|$	$\|\|\pi^-\|\| = (\|\|\pi\|\|)^{-1}$
$\|\|\langle\pi\rangle\varphi\|\| = \{V : \exists V_1 \text{ s.t. } \langle V, V_1\rangle \in \|\|\pi\|\| \text{ and } V_1 \in \|\|\varphi\|\|\}$	$\|\|\varphi?\|\| = \{\langle V, V\rangle : V \in \|\|\varphi\|\|\}$

Proposition 2. *Suppose $\mathbb{P}_\varphi \cap P = \emptyset$, i.e., none of the variables in P occurs in φ. Then $V \cup P \in \|\|\varphi\|\|$ iff $V \setminus P \in \|\|\varphi\|\|$.*

A distinguishing feature of DL-PA is that its dynamic operators can be eliminated (which is impossible in PDL). Just as for QBF, the resulting formula may be exponentially longer than the original formula.

Theorem 1 ([BHT13]). *For every* DL-PA *formula there is an equivalent boolean formula.*

Every assignment sequence $\alpha_1;\cdots;\alpha_n$ is a deterministic program that is always executable: for a given V, there is exactly one V' such that $\langle V, V'\rangle \in \|\|\alpha_1;\cdots;\alpha_n\|\|$. Moreover, when a set of update actions $\{\alpha_1,\ldots,\alpha_n\}$ is consistent then the order of the α_i in a sequential composition is irrelevant. The following can be viewed as a reformulation of Proposition 1 in terms of the DL-PA operator of sequential composition.

Proposition 3. *Let $\{\alpha_1,\ldots,\alpha_n\}$ be a consistent set of update actions. Let $\langle k_1 \ldots k_n\rangle$ be some permutation of $\langle 1 \ldots n\rangle$. Then $V \circ \{\alpha_{k_1},\ldots,\alpha_{k_n}\}$ equals the single V' such that $\langle V, V'\rangle \in \|\|\alpha_{k_1};\cdots;\alpha_{k_n}\|\|$.*

This entitles us to use sets of consistent update actions as programs: one may suppose that this stands for a sequential composition in some predefined order (based e.g. on the enumeration of the set of propositional variables).

4 Static Constraints and the Associated Repairs

In this section we consider the classical notion of database integrity that is defined in terms of *static integrity constraints* (or *static constraints* for short). In our propositional language they are nothing but boolean formulas. Two ways of repairing databases can be found in the literature on active integrity constraints [CT11]. Both consist in first finding an appropriate set of update actions U and then building the update $V \circ U$ of V by U as defined in Section 2. We relate them to well-known operations in belief revision and update [KM92], which allows us to reuse their embeddings into DL-PA [Her14].

4.1 Weak Repairs and Drastic Updates

Let V be a database and let C be a set of static constraints. A *weak repair* of V achieving C is a consistent set of update actions $U \subseteq \mathbb{U}$ such that $V \circ U \models \bigwedge C$ and such that U is *relevant* w.r.t. V. The latter means that $p\leftarrow\top \in U$ implies $p \notin V$ and $p\leftarrow\bot \in U$ implies $p \in V$.

Example 1. Let $V = \emptyset$ and let $C = \{p \vee q\}$. The weak repairs of V achieving C are all those subsets of the set of positive update actions $\{r\leftarrow\top \ : \ r \in \mathbb{P}\}$ that contain either $p\leftarrow\top$, or $q\leftarrow\top$, or both.

The example illustrates that weak repairs are indeed very weak. As the following result shows, if we consider what is true in all possible weak repairs then we obtain what is called a drastic update in the literature on belief revision and update.[1]

Proposition 4. *Let V be a database and let C be a set of static constraints. Then*

$$\{V \circ U \ : \ U \text{ is a weak repair of } V \text{ achieving } C\} = \Big\| \bigwedge C \Big\|.$$

Note that a weak repair may contain assignments of variables that do not occur in C. To remedy this we define a *relevant weak repair* to be a weak repair U such that if $p\leftarrow\top$ or $p\leftarrow\bot$ occurs in U then $p \in \mathbb{P}_C$.

This corresponds to a very basic update semantics that is sometimes called Winslett's standard semantics [Win90].

4.2 Repairs *Tout Court* and Their Relation to Winslett's PMA

A *repair* of V achieving C is a weak repair of V achieving C that is minimal w.r.t. set inclusion: there is no weak repair of V achieving C that is strictly contained in it.

Example 2. Let $V = \emptyset$ and $C = \{p \vee q\}$. There are exactly two repairs of V achieving C, viz. $\{p\leftarrow\top\}$ and $\{q\leftarrow\top\}$.

We are now going to relate repairs to Winslett's possible models approach PMA [Win88, Win90]. Remember that the update of a database V by a boolean formula A according to the PMA is the set of V' such that $V' \models A$ and such that the symmetric difference between V and V' is minimal w.r.t. set inclusion. Formally, symmetric difference is defined as $\mathsf{D}(V, V') = \{p \ : \ V(p) \neq V'(p)\}$ and the PMA update of V by A is

$$V \diamond^{\mathsf{pma}} A = \{V' \ : \ V' \models A \text{ and there is no } V'' \in \|A\| \text{ such that } \mathsf{D}(V, V'') \subset \mathsf{D}(V, V')\}$$

For example, $\emptyset \diamond^{\mathsf{pma}} p \vee q = \{\{p\}, \{q\}\}$ and $\emptyset \diamond^{\mathsf{pma}} (p \wedge q) \vee r = \{\{p, q\}, \{r\}\}$.

Proposition 5. *Let V be a database and let C be a set of static constraints. Then*

$$\{V \circ U \ : \ U \text{ is a repair of } V \text{ by } C\} = V \diamond^{\mathsf{pma}} \Big(\bigwedge C \Big).$$

The above result justifies the term *PMA repair* that we are going to employ henceforth (because the mere term 'repairs' might lead to confusions).

[1] It is actually also a drastic revision because V is a complete database and update and revision coincide in that case [PNP+96].

4.3 Repairs and Weak Repairs in DL-PA

We now embed Winslett's standard semantics (and thereby relevant weak repairs) and the PMA (and thereby repairs *tout court*) into DL-PA. This was already done in [Her14], but our embeddings are slightly more elegant and are presented in a more uniform and streamlined way. We start with some auxiliary definitions.

To each propositional variable p we associate a *fresh* propositional variable $p^{\pm}$. Each proposition $p^{\pm}$ will register whether or not the proposition p has been modified along the update. This is necessary to ensure that every proposition is modified at most once during a repair. We extend the definition to sets of variables $P \subseteq \mathbb{P}$: $P^{\pm} = \{p^{\pm} \mid p \in P\}$.

First, we need a program that sets all the propositions in a given set P to $\bot$: $P{\leftarrow}\bot$ is the sequence of assignments $p{\leftarrow}\bot$ for all $p \in P$ (whose order does not matter, cf. Proposition 3). Therefore ${\mathbb{P}_C}^{\pm}{\leftarrow}\bot$ is going to initialise the relevant $p^{\pm}$ before the program containing $\mathsf{toggle}(p)$ below is executed.

Second, the following two DL-PA programs (1) modify a single proposition and store this and (2) undo that modification:

$$\begin{aligned} \mathsf{toggle}(p) &= \textbf{if } \neg p^{\pm} \textbf{ then } p{\leftarrow}\neg p;\, p^{\pm}{\leftarrow}\top \textbf{ else fail} &&= \neg p^{\pm}?;\, p{\leftarrow}\neg p;\, p^{\pm}{\leftarrow}\top \\ \mathsf{undo}(p) &= \textbf{if } p^{\pm} \textbf{ then } p{\leftarrow}\neg p;\, p^{\pm}{\leftarrow}\bot \textbf{ else fail} &&= p^{\pm}?;\, p{\leftarrow}\neg p;\, p^{\pm}{\leftarrow}\bot \end{aligned}$$

The idea is that the variable $p^{\pm}$ keeps track of the modifications of p: we are going to ensure that it is true only once p has been modified during the current update. Then $\mathsf{toggle}(p)$ will flip the truth value of p if this value has not been modified yet and records the modification by setting $p^{\pm}$ to $\top$; if p has already been made true then $\mathsf{toggle}(p)$ fails. The program $\mathsf{undo}(p)$ undoes this.

Then a weak repair that is relevant w.r.t. C is achieved by the following DL-PA program:

$$\mathsf{weakRepair}(C) = {\mathbb{P}_C}^{\pm}{\leftarrow}\bot; \left(\bigcup_{p \in \mathbb{P}_C} \mathsf{toggle}(p)\right)^*; \left(\bigwedge C\right)?$$

We note that since each variable can be updated at most once and since the order of the updates does not matter, this can be rewritten without the Kleene star as a sequence $(\mathsf{toggle}(p_1) \cup \textbf{skip}); \ldots; (\mathsf{toggle}(p_k) \cup \textbf{skip})$ where $p_1, \ldots, p_k$ are the variables in $\mathbb{P}_C$.

We finally define the following DL-PA formula:

$$\mathsf{Minimal}(C) = \neg \left\langle \bigcup_{p \in \mathbb{P}_C} \mathsf{undo}(p); \left(\bigcup_{p \in \mathbb{P}_C} \mathsf{undo}(p)\right)^* \right\rangle \bigwedge C$$

The program in this formula undoes a nonempty set of $\mathsf{toggle}(p)$ actions (and nondeterministically so, failing when there was no change at all). Therefore the formula $\mathsf{Minimal}(C)$ says that there is no execution of that program leading to a database closer to the actual database that satisfies the constraints. So the actual database corresponds to a minimal change of the initial database.[2]

[2] The difference with [Her14] is that our programs memorise that a variable has been flipped instead of storing its previous value.

Theorem 2. *Let C be a set of static constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database (i.e., no $p^{\pm}$ occurs in either of them). Let U be a consistent set of update actions that is relevant w.r.t. V. Set $V' = (V \circ U) \cup \{p^{\pm} \ : \ p \leftarrow \top \in U \text{ or } p \leftarrow \bot \in U\}$.*

- *U is a relevant weak repair of V achieving C if and only if $\langle V, V' \rangle \in \|\mathsf{weakRepair}(C)\|$.*
- *U is a PMA repair of V achieving C iff $\langle V, V' \rangle \in \|\mathsf{weakRepair}(C); \mathsf{Minimal}(C)?\|$.*

Proof. For the first item, observe that $\langle V, V' \rangle \in \|\mathsf{weakRepair}(C)\|$ if and only if $V' \in \|C\|$ and the following holds for all variables $p \in \mathbb{P}$ (i.e., excluding the $p^{\pm}$): (a) $p^{\pm} \in V'$ iff $V(p) \neq V'(p)$ and (b) if $V(p) \neq V'(p)$ then $p \in \mathbb{P}_C$, i.e., only p's from C and the associated $p^{\pm}$ were modified.

For the second item, given some actual database V', define the initial database as

$$V = \{p \in \mathbb{P} \ : \ p \in V' \text{ and } p^{\pm} \notin V'\} \cup \{p \in \mathbb{P} \ : \ p \notin V' \text{ and } p^{\pm} \in V'\}.$$

Then $V' \in \|\mathsf{Minimal}(C)\|$ iff there is no $V'' \in \| \bigwedge C\|$ such that $\mathsf{D}(V, V'') \subset \mathsf{D}(V, V')$.[3]

5 Active Constraints and the Associated Repairs

Active integrity constraints were proposed about ten years ago [FGZ04], and various ways of repairing a database V by such constraints have been studied in the literature. We refer to [CT11] for an overview. Just as for static constraints, all definitions are based on the notion of *repair set*: an appropriate set of update actions U such that $V \circ U$ no longer violates the integrity constraints, where $V \circ U$ is the result of updating V with U as defined in Section 2 and is called the *repaired database*.

In the present section we recall syntax and semantics and show that they can be recast in DL-PA.

5.1 Active Integrity Constraints

An *active integrity constraint* (or *active constraint* for short), combines a static integrity constraint with a preferred repair action. Formally, an active constraint is a couple

$$r = \langle \mathsf{C}(r), \mathsf{R}(r) \rangle$$

where $\mathsf{C}(r)$ is a boolean formula and $\mathsf{R}(r)$ is a finite set of update actions that is consistent. As before, $\mathsf{C}(r)$ is a static integrity constraint that is violated when $\mathsf{C}(r)$ is false. If so then r is *applicable* and $\mathsf{R}(r)$ indicates how to get rid of the violation and achieve integrity. We view the elements of $\mathsf{R}(r)$ as *permitted* update actions: When $\mathsf{C}(r)$ is violated then each of the actions in $\mathsf{R}(r)$ gets a 'license to update'.[4] This is a rather imprecise description of the job the update actions in $\mathsf{R}(r)$ are expected to do, and in the literature various semantics are associated to a set of active constraints. For one of the most

[3] Note that by definition of $\mathsf{toggle}(p)$, $p \in \mathsf{D}(V, V')$ is equivalent to $p^{\pm} \in \mathsf{D}(V, V')$ thus the inclusion $\mathsf{D}(V, V'') \subset \mathsf{D}(V, V')$ is not affected by the variables in ${\mathbb{P}_C}^{\pm}$.

[4] The reading that is given in the literature is slightly different from ours: there, $\mathsf{R}(r)$ is called the set of preferred update actions.

prominent of them in terms of founded repairs, it turns out that the elements of $\mathsf{R}(r)$ have to be viewed as *exclusive choices*: when some $\alpha \in \mathsf{R}(r)$ is part of the repair set then no other β can be part of the repair set.

We say that an active constraint $r = \langle \mathsf{C}(r), \mathsf{R}(r) \rangle$ is *standard* if $\mathsf{C}(r)$ is a clause and each update action in $\mathsf{R}(r)$ produces one of the literals of $\mathsf{C}(r)$: if $p \leftarrow \top \in \mathsf{R}(r)$ then p has to be one of the literals of $\mathsf{C}(r)$ and if $p \leftarrow \bot \in \mathsf{R}(r)$ then $\neg p$ has to be one of the literals of $\mathsf{C}(r)$.

Remark 1. The definition in the literature differs in several respects from ours here. First, $\mathsf{C}(r)$ is not viewed as a static integrity constraint but as the negation of a static integrity constraint (r is violated when the first argument of r is true). Second, active constraints are noted $\mathsf{C}(r) \rightarrow \mathsf{R}(r)$, which makes them look like formulas. However, such formulas are non-standard because the right hand side of the implication is not a formula but a set of programs. So their semantics remains to be given: in the literature this is typically done by means of disjunctive logic programs under a non-monotonic semantics. Third, all active constraints have to be standard.

We denote finite sets of active constraints by η, η_1, etc. The set of static integrity constraints associated to such a set is defined as $\mathsf{C}(\eta) = \{\mathsf{C}(r) \ : \ r \in \eta\}$.

It remains to associate a semantics to active constraints. In the present and the following section we discuss the options and their properties.

5.2 Founded Weak Repairs and Founded Repairs

In the literature, founded repairs are considered to be a natural basic semantics of active constraints that is a good starting point for further refinements.

Given a set of active constraints η and a database V, a consistent set of update actions U is *founded* if for every $\alpha \in U$ there is an $r \in \eta$ such that (a) $\alpha \in \mathsf{R}(r)$, (b) $V \circ U \models \mathsf{C}(r)$, and (c) $V \circ (U \setminus \{\alpha\}) \not\models \mathsf{C}(r)$. A set of update actions U is a *founded (weak) repair* of V by η if U is a (weak) repair of V achieving $\mathsf{C}(\eta)$ and U is founded.

Remark 2. We have reformulated the original definition so that it applies to our more general definition of active constraint. Both are equivalent as far as standard active constraints are concerned.

Founded repairs do not necessarily exist [CT11, Example 2].

Example 3. Consider $\eta = \{\langle p, \{p \leftarrow \top\}\rangle, \langle p \vee q, \{q \leftarrow \top\}\rangle\}$. The set $\{p \leftarrow \top\}$ is a founded weak repair of $V_0 = \emptyset$ by η. It is the only such repair: the second update action in $\{p \leftarrow \top, q \leftarrow \top\}$ cannot be founded on the second active constraint of η.

In the next section, we propose an encoding of the notion of founded repairs in DL-PA.

Example 4 ([CT11], Example 3). Consider

$$\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{p \leftarrow \top\}\rangle, \langle p \vee \neg q, \{q \leftarrow \top\}\rangle\}.$$

The set $\{p \leftarrow \top, q \leftarrow \top\}$ is the only founded repair of $V_0 = \emptyset$ by η.

This illustrates *circularity of support*: each update action is individually founded because the others happen to be in the repair. Such repairs are considered to be unintended and the notion of *justified repair* was proposed to overcome the problem. Justified repairs can be encoded in DL-PA in a way similar to the encoding of founded repairs. We however do not work this out here.

5.3 Founded Repairs in DL-PA

We re-use the abbreviations $\mathsf{weakRepair}(\mathsf{C}(\eta))$ and $\mathsf{Minimal}(\mathsf{C}(\eta))$ that we have introduced in Section 4.3. Remember that in order to keep track of modifications we had supposed that we have at our disposal fresh variables $p^{\pm}$, one per variable $p \in \mathbb{P}$. We moreover need the following:

$$\mathsf{IsFounded}(\eta) = \bigwedge_{p \in \mathbb{P}_{\mathsf{C}(\eta)}} \left(p^{\pm} \to \bigvee_{\substack{r \in \eta \\ p \leftarrow \mathsf{X} \in \mathsf{R}(r)}} \langle p \leftarrow \neg p \rangle \neg \mathsf{C}(r) \right)$$

where X ranges over $\{\top, \bot\}$. The formula is true if and only if all current update actions (encoded in the current valuation by means of the fresh variables $p^{\pm}$) are founded.

Theorem 3. *Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V_0 \subseteq \mathbb{P}$ be a database (i.e., no $p^{\pm}$ occurs in either of them). Let U be a consistent set of update actions that is relevant w.r.t. V_0.*

- *U is a weak founded repair of V_0 by η iff*

$$\langle V_0, V_0 \circ U \rangle \in \left\| \mathsf{weakRepair}(\mathsf{C}(\eta)); \mathsf{IsFounded}(\eta)? \right\|.$$

- *U is a founded repair of V_0 by η iff*

$$\langle V_0, V_0 \circ U \rangle \in \left\| \mathsf{weakRepair}(\mathsf{C}(\eta)); \mathsf{IsFounded}(\eta)?; \mathsf{Minimal}(\mathsf{C}(\eta))? \right\|.$$

Proof. Suppose V is some repaired database (containing variables $p^{\pm}$). Define the set of update actions

$$U_{V,\eta} = \{p \leftarrow \top \ : \ p^{\pm} \in V \text{ and } p \in V\} \cup \{p \leftarrow \bot \ : \ p^{\pm} \in V \text{ and } p \notin V\}.$$

Let us prove that $V \in \|\mathsf{weakRepair}(\mathsf{C}(\eta)); \mathsf{IsFounded}(\eta)\|?$ iff $U_{V,\eta}$ is a weak founded repair of V_0 by η. The latter means that for every $\alpha \in U_{V,\eta}$, the three conditions (a) $\alpha \in \mathsf{R}(r)$, (b) $V_0 \circ U_{V,\eta} \models \mathsf{C}(r)$, and (c) $V_0 \circ (U_{V,\eta} \setminus \{\alpha\}) \not\models \mathsf{C}(r)$ are satisfied.

For the left-to-right direction consider some $p \leftarrow \top \in U_{V,\eta}$. Then $p^{\pm} \in V$. Condition (b) is satisfied from the definition $\mathsf{weakRepair}(\mathsf{C}(\eta))$ and Theorem 2. Condition (a) is satisfied by the existence of a candidate rule in the definition of $\mathsf{IsFounded}(\eta)$; remark that we are guaranteed that the rule contains indeed $p \leftarrow \top$, as opposed to $p \leftarrow \bot$, because undoing the change on p changes $\mathsf{C}(r)$ to false (so X has to be $\top$). Condition (c) is satisfied because $V_0 \circ (U \setminus \{p \leftarrow \top\}) \not\models \mathsf{C}(r)$ is equivalent to $V_0 \circ U \models \neg \langle p \leftarrow \bot \rangle \mathsf{C}(r)$.

For the right-to-left direction, Theorem 2 ensures that $U_{V,\eta}$ is a weak repair. To prove that $V \in \|\mathsf{IsFounded}(\eta)\|$, consider some $p^{\pm} \in V$. By definition, it entails $p \leftarrow \mathsf{X} \in U_{V,\eta}$ for some $X \in \{\top, \bot\}$. Condition (a) ensures that there is a rule $r \in \eta$ with $p \leftarrow \top \in \mathsf{R}(r)$. Condition (c) implies $V \models \neg \langle p \leftarrow \neg X \rangle \mathsf{C}(r)$. This concludes.

6 A New Definition of Repair in DL-PA

We now propose two new definitions that take advantage of the resources of DL-PA. More precisely, we make use of **while** loops in order to iterate the application of active constraints. We start by discussing how databases can be repaired by applying active constraints in sequence. This will lead us to the definition of dynamic repair. We show that it is incomparable with both founded weak repairs and founded repairs.

6.1 Repairing a Database: A Dynamic View

Suppose there is only one active constraint r that is standard. Then it is clear how to proceed: either $V \models \mathsf{C}(r)$ and there is nothing to do, or $V \not\models \mathsf{C}(r)$ and we have to apply r. In the second case, each $\alpha_i \in \mathsf{R}(r)$ provides a PMA repair of V achieving $\mathsf{C}(r)$.[5] What about the case where $\mathsf{R}(r)$ is empty? Well, then V cannot be repaired and we are stuck.

So far so good. The situation gets way more intricate when the set of active constraints η contains two or more elements that can interact.

Even for standard active constraints it might not be enough to apply only one of the update actions from $\mathsf{R}(r)$: some of the active constraints might have to be applied several times in order to obtain integrity. The following example of an n-bit counter highlights this.

Example 5. Suppose we represent binary numbers up to $2^{n+1}-1$ by means of $n+1$ propositional variables: $\neg p_n \wedge \cdots \wedge \neg p_0$ represents the integer zero and $p_n \wedge \cdots \wedge p_0$ represents $2^{n+1}-1$. Let

$$
\begin{aligned}
r_1 &= \langle p_0 \vee x_0 \vee \cdots \vee x_n, \{p_0 \leftarrow \top\}\rangle \\
r_{2_k} &= \langle p_k \vee \neg p_{k-1} \vee \cdots \vee \neg p_0 \vee x_k, \{x_k \leftarrow \top\}\rangle, \quad \text{for } k \leq n \\
r_{3_k} &= \langle p_k \vee \neg p_{k-1} \vee \cdots \vee \neg p_0 \vee \neg x_k, \{p_k \leftarrow \top, p_{k-1} \leftarrow \bot, \ldots, p_0 \leftarrow \bot\}\rangle, \quad \text{for } k \leq n \\
r_{4_k} &= \langle \neg p_k \vee p_{k-1} \vee \cdots \vee p_0 \vee \neg x_k, \{x_k \leftarrow \bot\}\rangle, \quad \text{for } k \leq n
\end{aligned}
$$

The idea is that when $\neg p_k \wedge p_{k-1} \wedge \cdots \wedge p_0$ is true, i.e., when the number $011\ldots1$ has to be incremented to $100\ldots0$, then x_k is made true by r_{2_k} and remains so unless $100\ldots0$ has been attained. This involves flipping the k digits in the conjunction $\neg p_k \wedge p_{k-1} \wedge \cdots \wedge p_0$: with active constraints this is done one-by-one by the rule r_{3_k}. Then x_k is set to false again by r_{4_k}. Let $\eta_n = \{r_1\} \cup \{r_{2_1}, \ldots, r_{2_n}\} \cup \{r_{3_1}, \ldots, r_{3_n}\} \cup \{r_{4_1}, \ldots, r_{4_n}\}\}$. Successive repairing steps implement an n-bit counter counting from the initial database $\emptyset$ to the database $\{p_n, \ldots, p_0\}$.

The computation takes $2^{n+1}-1$ steps, demonstrating that sometimes atomic repairs must be performed an exponential number of times: $V_0 = \emptyset$ can only be repaired by applying r_1 a number of times exponential in n.

Our example highlights the difference between dynamic repairs and founded repairs: in the latter an active constraint can only be used once.

[5] For our more general active constraints where there is no syntactical link between $\mathsf{C}(r)$ and $\mathsf{R}(r)$ we have to compute all possible minimal subsets $U \subseteq \mathsf{R}(r)$ such that $V \models \mathsf{C}(r)$. All of them are PMA repairs.

6.2 Dynamic Weak Repairs and Dynamic Repairs

We associate to every active constraint r the DL-PA programs

$$\pi_r = \neg \mathsf{C}(r)?; \bigcup_{\alpha \in \mathsf{R}(r)} \alpha \quad \text{and} \quad \pi_r^{\pm} = \neg \mathsf{C}(r)?; \bigcup_{p \leftarrow \mathsf{X} \in \mathsf{R}(r)} (p \leftarrow \mathsf{X}; p^{\pm} \leftarrow \top),$$

where we consider that $\bigcup_{\alpha \in \mathsf{R}(r)} \alpha$ equals **fail** when $\mathsf{R}(r)$ is empty. This matches the intuitive reading that we have given to active constraints in Section 5.1: the repair program π_r checks whether the static integrity constraint associated to r is violated and if so applies one of the update actions from $\mathsf{R}(r)$. The program $\pi_r^{\pm}$ moreover stores that p has been changed. This is also supported by the following proposition, which tells us that applicability of an active constraint r is matched by the DL-PA notion of executability of the program π_r.

Proposition 6. *Let r be an active constraint and let V be a database. Then applicability of r at V is equivalent to both $V \models \langle \pi_r \rangle \top$ and $V \models \langle \pi_r \rangle^{\pm} \top$.*

Proof. It suffices to observe that when π is a nondeterministic composition of update actions then the equivalence $\varphi \leftrightarrow \langle \varphi?; \pi \rangle \top$ is DL-PA valid for every φ.

A *dynamic weak repair* of V by η is a set of update actions U such that U is relevant w.r.t. V and

$$\langle V, V \circ U \rangle \in \Big\| \textbf{while } \neg \mathsf{C}(\eta) \textbf{ do } \Big(\bigcup_{r \in \eta} \pi_r \Big) \Big\|.$$

Finally, U is a *dynamic repair* of V by η if U is a PMA repair of V by η that is dynamic.

Example 6 (Example 4, ctd.). Consider again

$$\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{p \leftarrow \top\}\rangle, \langle p \vee \neg q, \{q \leftarrow \top\}\rangle\}.$$

There is a single dynamic (weak) repair of $V_0 = \emptyset$ by η, viz. $\{p \leftarrow \top, q \leftarrow \top\}$.

Example 7 (Example 3, ctd.). Consider again $\eta = \{\langle p, \{p \leftarrow \top\}\rangle, \langle p \vee q, \{q \leftarrow \top\}\rangle\}$, whose only founded weak repair was $\{p \leftarrow \top\}$. There are two dynamic weak repairs of $V_0 = \emptyset$ by η, namely $\{p \leftarrow \top\}$ and $\{p \leftarrow \top, q \leftarrow \top\}$. Only the former is a dynamic repair.

The next example illustrates that dynamic weak repairs are not necessarily founded.

Example 8. Consider $\eta = \{\langle p \vee q, \{p \leftarrow \top, q \leftarrow \top\}\rangle, \langle p \vee r, \{p \leftarrow \top, r \leftarrow \top\}\rangle\}$. There are four dynamic weak repairs of $V_0 = \emptyset$ by η, namely $U_1 = \{p \leftarrow \top\}$, $U_2 = \{q \leftarrow \top, r \leftarrow \top\}$, $U_1' = \{p \leftarrow \top, q \leftarrow \top\}$, and $U_1'' = \{p \leftarrow \top, r \leftarrow \top\}$. Only U_1 and U_2 are dynamic repairs.

The next theorem characterises dynamic repairs in terms of DL-PA programs.

Theorem 4. *Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V_0 \subseteq \mathbb{P}$ be a database (i.e., no $p^{\pm}$ occurs in either of them). Let U be a consistent set of update actions that is relevant w.r.t. V_0. U is a dynamic repair of V_0 by η iff*

$$\langle V_0, V_0 \circ U \rangle \in \Big\| \textbf{while } \neg \mathsf{C}(\eta) \textbf{ do } \Big(\bigcup_{r \in \eta} \pi_r^{\pm} \Big); \mathsf{Minimal}\big(\mathsf{C}(\eta)\big)? \Big\|.$$

Other definitions of dynamic repairs are possible. We could e.g. stipulate that U is a dynamic repair of V if it is a dynamic weak repair that is minimal w.r.t. set inclusion, i.e., such that there is no dynamic weak repair U' of V such that $U' \subset U$. We have not explored this option in detail, but it seems that it can be captured in DL-PA as well.

7 Discussion and conclusion

We have shown how several definitions of database repair via active integrity constraints can be expressed in DL-PA, including a new proposal in terms of their iterated application. This allows us to claim that DL-PA is a nice integrated framework for database updates: it not only provides operators $p{\leftarrow}\top$ of insertion and $p{\leftarrow}\bot$ of deletion and more generally sets U of such assignments that can be applied to a database V; it also provides a means to reason about the repair of the resulting $V{\circ}U$ when some element of the set of integrity constraints is violated. For example, V' is a possible repair of the update of the database V by the deletion of p if and only if the couple $\langle V, V'\rangle$ belongs to the interpretation of the DL-PA program $p{\leftarrow}\bot; \mathit{repair}$, where *repair* is one of the repair programs of theorems 2, 3, 4. Moreover, the set of candidate repaired databases is the interpretation of the DL-PA formula $\langle(p{\leftarrow}\bot; \mathit{repair})^{-}\rangle\varphi_V$, where φ_V is a conjunction of literals describing V syntactically.

Beyond identifying possible repaired databases, our programs *repair* also allow to solve decision problems. For example, we may check whether it is possible at all to repair V by model checking in DL-PA whether

$$V \models \langle \mathit{repair}\rangle\top.$$

We can also check whether there is a unique repair of V by model checking whether the set of databases V' such that $\langle V, V'\rangle \in \|\mathit{repair}\|$ is a singleton. This amounts to model check for each of the variables p occurring in the constraints whether

$$V \models [\mathit{repair}]p \vee [\mathit{repair}]\neg p.$$

We might as well wish to check possibility or unicity of the repairs independently of a specific database V. For example, we can check whether η can repair any database by checking whether the formula $\langle \mathit{repair}\rangle\top$ is DL-PA valid. A further interesting reasoning task is to check whether two sets of active constraints η_1 and η_2 are equivalent under a given semantics by checking whether $\|\mathit{repair}_{\eta_1}\| = \|\mathit{repair}_{\eta_2}\|$.

Our active integrity programs of the form $r = \langle \mathsf{C}(r), \mathsf{R}(r)\rangle$ generalise the condition $\mathsf{C}(r)$ from disjunctions of clauses to arbitrary formulas (that could actually even be DL-PA formulas). This opens up two perspectives. First, our definition also covers revision programs [CT11]; we leave it to future work to establish the exact relationship. Second, we could further generalise the action $\mathsf{R}(r)$ from a set of update actions to arbitrary DL-PA programs. Dynamic repairs would then still make sense, while it is not clear how founded and justified repairs would have to be defined.

It is known that deciding the existence of a repair is NP-complete for PMA repairs and for founded weak repairs, while it is Σ_P^2 complete for founded repairs [CT11]. We leave to future work the investigation of the complexity of dynamic repairs. What can already be said is that our repair programs repair_η all have length polynomial in the size of η. Complexity results for the fragments of DL-PA containing the respective repair programs would therefore provide an upper complexity bound. These results remain to be established; they would parallel those for fragments of QBF. First steps are in [Her14].

Acknowledgements. Thanks are due to the reviewers for helpful comments.

References

[BHT13] Balbiani, P., Herzig, A., Troquard, N.: Dynamic logic of propositional assignments: a well-behaved variant of PDL. In: Kupferman, O. (ed.) Logic in Computer Science (LICS). IEEE (2013)

[CF14] Cruz-Filipe, L.: Optimizing computation of repairs from active integrity constraints. In: Beierle, C., Meghini, C. (eds.) FoIKS 2014. LNCS, vol. 8367, pp. 361–380. Springer, Heidelberg (2014)

[CGZ09] Caroprese, L., Greco, S., Zumpano, E.: Active integrity constraints for database consistency maintenance. IEEE Trans. Knowl. Data Eng. 21(7), 1042–1058 (2009)

[CT08] Caroprese, L., Truszczyński, M.: Declarative semantics for active integrity constraints. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 269–283. Springer, Heidelberg (2008)

[CT11] Caroprese, L., Truszczynski, M.: Active integrity constraints and revision programming. TPLP 11(6), 905–952 (2011)

[CTZ07] Caroprese, L., Trubitsyna, I., Zumpano, E.: View updating through active integrity constraints. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 430–431. Springer, Heidelberg (2007)

[DHP14] Doutre, S., Herzig, A., Perrussel, L.: A dynamic logic framework for abstract argumentation. In: International Conference on Principles of Knowledge Representation and Reasoning (KR), Vienna, Austria, pp. 143–152. AAAI Press (2014)

[FGZ04] Flesca, S., Greco, S., Zumpano, E.: Active integrity constraints. In: Moggi, E., Warren, D.S. (eds.) PPDP, pp. 98–107. ACM (2004)

[Har84] Harel, D.: Dynamic logic. In: Gabbay, D.M., Günthner, F. (eds.) Handbook of Philosophical Logic, vol. II, pp. 497–604. D. Reidel, Dordrecht (1984)

[Her14] Herzig, A.: Belief change operations: A short history of nearly everything, told in dynamic logic of propositional assignments. In: Baral, C., De Giacomo, G. (eds.) Proc. KR 2014. AAAI Press (2014)

[HKT00] Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)

[HLMT11] Herzig, A., Lorini, E., Moisan, F., Troquard, N.: A dynamic logic of normative systems. In: Walsh, T. (ed.) International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, pp. 228–233. IJCAI/AAAI (2011)

[HMNDBW14] Herzig, A., Menezes, V., De Barros, L.N., Wassermann, R.: On the revision of planning tasks. In: Schaub, T. (ed.) European Conference on Artificial Intelligence (ECAI) (August 2014)

[KM92] Katsuno, H., Mendelzon, A.O.: On the difference between updating a knowledge base and revising it. In: Gärdenfors, P. (ed.) Belief Revision, pp. 183–203. Cambridge University Press (1992); preliminary version in Allen, J.A., Fikes, R., Sandewall, E. (eds.): Principles of Knowledge Representation and Reasoning: Proc. 2nd Int. Conf., pp. 387–394. Morgan Kaufmann Publishers (1991)

[PNP+96] Peppas, P., Nayak, A.C., Pagnucco, M., Foo, N.Y., Kwok, R.B.H., Prokopenko, M.: Revision vs. update: Taking a closer look. In: Wahlster, W. (ed.) ECAI, pp. 95–99. John Wiley and Sons, Chichester (1996)

[TM85] Tiomkin, M.L., Makowsky, J.A.: Propositional dynamic logic with local assignments. Theor. Comput. Sci. 36, 71–87 (1985)

[vE00] van Eijck, J.: Making things happen. Studia Logica 66(1), 41–58 (2000)

[Win88] Katsuno, H., Mendelzon, A.O.: On the difference between updating a knowledge base and revising it. In: Gärdenfors, P. (ed.) Belief Revision, pp. 183–203. Cambridge University Press (1992); Reasoning about action using a possible models approach. In: Proc. 7th Conf. on Artificial Intelligence (AAAI 1988), St. Paul, pp. 89–93 (1988)

[Win90] Winslett, M.-A.: Updating Logical Databases. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (1990)